

Towards Nominal Formal Languages (long version)

Alexander Kurz, Tomoyuki Suzuki*, and Emilio Tuosto

Department of Computer Science, University of Leicester, UK

Abstract. We introduce formal languages over infinite alphabets where words may contain binders. We define the notions of nominal language, nominal monoid, and nominal regular expressions. Moreover, we extend history-dependent automata (HD-automata) by adding stack, and study the recognisability of nominal languages.

1 Introduction

Automata over infinite alphabets have been receiving an increasing amount of attention, see eg [11, 15, 1, 17]. In these approaches, the countably infinite alphabet \mathcal{N} can be considered as a set of ‘names’, which can be tested only for equality. Typically, languages of interest such as

$$\mathcal{L}_1 = \{n_1 \dots n_k \in \mathcal{N}^* \mid \exists i \neq j. n_i = n_j\} \quad (1)$$

from [11] are invariant under name-permutations: If eg nmn is in the language, then so is $n'mn' = (n \ n') \cdot nmn$, where $(n \ n') \cdot nmn$ stands for the application of the transposition $(n \ n')$ to the word nmn . This suggests to think of the names as being bound and languages to be closed under α -equivalence. On the other hand, we may fix a name n_1 and consider the language

$$\mathcal{L}_{2,n_1} = \{n_1 n_2 \dots n_k \in \mathcal{N}^* \mid \forall i \neq j. n_i \neq n_j\} \quad (2)$$

from [17]; we can think of n_1 as a free name and of the n_2, \dots, n_k as bound. This suggests to study not only words over names, but also words which contain binders and allow us to make explicit the distinction between bound and free names. Automata on words with binders already appear in [16] in the study of the λ -calculus. *In this paper we begin the systematic study of words with binders from the point of view of the classical theory of formal languages and automata.*

In particular, our contributions are:

- *nominal languages* of words with binders (§ 2) as a natural generalisation of formal languages over infinite alphabets;
- *nominal monoids* (§ 3) as the corresponding algebraic structures;
- *nominal regular expressions* (§ 4) as a generalisation of regular expressions;

* The author’s PhD research is supported by Yoshida Scholarship Foundation.

- *HD-automata with stack* (HDS) (§ 5) and Theorem 2 showing that nominal regular expressions can be faithfully encoded into HDS.

One of the motivations to study words with binders comes from verification. For instance, consider the Needham-Schroeder protocol

$$\begin{aligned} A &\rightarrow B : \{n, A\}_B \\ B &\rightarrow A : \{n, m\}_A \\ A &\rightarrow B : \{m\}_B \end{aligned}$$

The (correct) runs of the protocol can be characterised by a nominal regular expression

$$\langle n. \text{ ENCR } n \text{ A FOR } B \ \langle m. \text{ ENCR } n \text{ m FOR } A \ (\text{ ENCR } m \text{ FOR } B) \rangle \rangle^* \quad (3)$$

where the alphabet is now $\mathcal{N} \cup \mathcal{S}$ with $n, m \in \mathcal{N}$ and $\{\text{ENCR}, \text{FOR}, A, B\} = \mathcal{S}$ a finite set of ‘letters’; finally, $\langle n.e \rangle$ binds all the free occurrences of n in e and *generates a fresh name* n . From (3) one could obtain an HDS for monitoring the execution of a protocol, i.e. the HDS would be able to detect if something goes wrong during the execution (e.g., an intruder is performing an attack). From an automata theoretic point of view, the interesting new feature appears more clearly if we abstract (3) to

$$\langle n.n \langle m.nm \rangle \rangle^* \quad (4)$$

and note that binding (fresh name generation) $\langle \dots \rangle$ appears under the Kleene star, which is the reason why automata accepting such languages need to have a stack.

2 Nominal Languages

We introduce languages with name binders. This section appeals to our intuitive understanding of binding and α -equivalence as known from eg λ -calculus or first-order logic, but see the next section for a formal treatment. To start with, the *alphabet* is divided disjointly into a countably infinite set \mathcal{N} (of *names*) and a finite set \mathcal{S} (of *letters*).

Definition 1 (m-word). An m-word is a term built from constants $\mathcal{N} \cup \mathcal{S} \cup \{\varepsilon\}$, and two binary operations $\circ, \langle \dots \rangle$, according to

$$w \stackrel{\text{def}}{=} \varepsilon \mid n \mid s \mid w \circ w \mid \langle n.w \rangle,$$

where n ranges over \mathcal{N} and s over \mathcal{S} . We denote by \mathbf{M} the set of all m-words.

As in the classical case we assume that ε (the empty word) is the neutral element wrt \circ and that \circ is associative. We often write wv for the concatenation $w \circ v$. Furthermore, we let $\langle n.w \rangle$ bind the free occurrences of n in w and take m-words up to α -equivalence.

The notion of m-word is the *most* general notion of word with binders: We only require from words to form a monoid and behave well wrt α -equivalence. Due to the scope introduced by binding, words now have a tree structure. This motivates the following, more special, but perhaps more naturally generalised, notion of words.

Definition 2 (g-word). A g-word is a term built from ε , unary operations n_{\cdot} , s_{\cdot} for each $n \in \mathcal{N}$, $s \in \mathcal{S}$, and a binary operation $\langle\langle \cdot, \cdot \rangle\rangle$, according to

$$w \stackrel{\text{def}}{=} \varepsilon \mid nw \mid sw \mid \langle\langle n.w \rangle\rangle.$$

We denote by \mathbf{G} the set of all g-words.

Regarding binding and α -equivalence, we follow the same conventions as for m-words. To consider \mathbf{G} as a monoid, we define $\circ: \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G}$ as follows:

$$\begin{aligned} \varepsilon \circ w &\stackrel{\text{def}}{=} w & nw \circ v &\stackrel{\text{def}}{=} n(w \circ v) \\ sw \circ v &\stackrel{\text{def}}{=} s(w \circ v) & \langle\langle n.w \rangle\rangle \circ v &\stackrel{\text{def}}{=} \langle\langle n'.(w' \circ v) \rangle\rangle \end{aligned} \quad (5)$$

where n' is fresh for v and $\langle\langle n'.w' \rangle\rangle$ is an α -renaming of $\langle\langle n.w \rangle\rangle$. Intuitively speaking, we extrude the scope of the binding to the end of the word.

Next we allow binders to appear only at the beginning of a word.

Definition 3 (l-word). An l-word is a pair (p, w) where $p \in \mathcal{N}^*$ and $w \in (\mathcal{N} \cup \mathcal{S})^*$. We denote by \mathbf{L} the set of all l-words.

We interpret p as a prefix of name binders and w as the part of the word that has no binders. $\circ: \mathbf{L} \times \mathbf{L} \rightarrow \mathbf{L}$ is given on the left below

$$(p, w) \circ (q, v) \stackrel{\text{def}}{=} (pq, wv) \quad [n](p, w) \stackrel{\text{def}}{=} (np, w) \quad (6)$$

where we assume that p and q , p and v , and q and w have no names in common. Whereas previously name-binding was built into the syntax via $\langle\langle \cdot, \cdot \rangle\rangle$, we now define explicitly, anticipating notation from § 3, a binding operation $[\cdot]: \mathcal{N} \times \mathbf{L} \rightarrow \mathbf{L}$ via the clause on the right of (6).

Definition 4 (s-word). An s-word is a pair (S, w) where $w \in (\mathcal{N} \cup \mathcal{S})^*$ and S is a subset of the names appearing in w . We denote by \mathbf{S} the set of all s-words.

On \mathbf{S} , we define the two operations \circ and $[\cdot]$ as follows, assuming that S and T , S and v , T and w have no names in common.

$$(S, w) \circ (T, v) \stackrel{\text{def}}{=} (S \cup T, wv) \quad [n](S, w) \stackrel{\text{def}}{=} \begin{cases} (S \cup \{n\}, w) & \text{if } n \text{ in } w \\ (S, w) & \text{otherwise} \end{cases} \quad (7)$$

Remark 1. We have embeddings $\mathbf{S} \xrightarrow{\text{sl}} \mathbf{L} \xrightarrow{\text{lg}} \mathbf{G} \xrightarrow{\text{gm}} \mathbf{M}$. For sl we assume that names are ordered; the other main clauses are $\text{lg}(np, w) = \langle\langle n.\text{lg}(p, w) \rangle\rangle$ and $\text{gm}(nw) = n \circ \text{gm}(w)$.

3 Nominal monoids

The somewhat informal treatment of § 2 should be sufficient to understand how automata process words with binders in § 5 and § 6. On the other hand, from a conceptual point of view, it is important to have a unifying account. The presence of names and

binders suggests to employ nominal sets [8]. This not only provides us with a mathematical theory, but also a clear conceptual guidance: Follow the classical universal algebraic account of languages and automata, but replace sets by nominal sets. Here, we apply this to languages and monoids.

Nominal sets and their logics come in different versions. We follow [9], for which we need to refer to for details. More details can also be found in [12]. Let us just recall

Definition 5 (Nominal set). Denote by $\text{Perm}(\mathcal{N})$ the group of permutations of \mathcal{N} generated from the set of transpositions $\{(n\ m) \mid n, m \in \mathcal{N}\}$. A set A equipped with a $\text{Perm}(\mathcal{N})$ -action $\text{Perm}(\mathcal{N}) \times A \rightarrow A$ is a nominal set, if every element in A is finitely supported. This means that for each $a \in A$ there is finite set $S \subseteq \mathcal{N}$ (called a support of a) such that $\pi|_S = \text{id} \Rightarrow \pi \cdot a = a$ for all $\pi \in \text{Perm}(\mathcal{N})$ (where $\pi|_S$ denotes the restriction of π to S). Maps between nominal sets are required to be equivariant, that is, they respect the permutation action.

It follows that each element $a \in A$ has a minimal support $\text{supp}(a)$ and one writes $\mathbf{n}\#a$ (n is fresh for a) for $n \notin \text{supp}(a)$. This allows us to define abstraction [8, Lemma 5.1] as $[n]a \stackrel{\text{def}}{=} \{(n, a)\} \cup \{(m, (nm) \cdot a) \mid m\#a\}$ and $[\mathcal{N}]A \stackrel{\text{def}}{=} \{[n]a \mid n \in \mathcal{N}, a \in A\}$.

A **nominal algebra** \mathfrak{A} , see [9, Def 4.13], consists of a nominal set A , constants $n \in \mathcal{N}$, and a map $[\mathcal{N}]A \rightarrow A$. As in universal algebra, further operations and equations may be added:

Definition 6. A nominal monoid is a nominal algebra \mathfrak{A} with additional constants $s \in \mathcal{S}$ and (equivariant) operations ε, \circ so that (A, ε, \circ) is a monoid.

We say that $w \in A$ is **closed**, or that w contains no free names, if $\text{supp}(w)$ is empty.

Definition 7. Write C_m for the class of all nominal monoids. We consider the following axioms where $m, n \in \mathcal{N}$, $s \in \mathcal{S}$, and X, Y are variables ranging over carriers of algebras.

$$\begin{array}{ll} \mathbf{Ax1} & n\#Y \vdash [n]X \circ Y = [n](X \circ Y) \\ \mathbf{Ax3} & n\#m \vdash n \circ [m]Y = [m](n \circ Y) \\ \mathbf{Ax5} & n\#X \vdash [n]X = X \end{array} \quad \begin{array}{ll} \mathbf{Ax2} & s \circ [m]Y = [m](s \circ Y) \\ \mathbf{Ax4} & \vdash [n][m]X = [m][n]X \\ \mathbf{Ax6} & n\#X \vdash X \circ [n]Y = [n](X \circ Y) \end{array}$$

C_g, C_l, C_s are axiomatised by Ax1, Ax1-3, Ax1-5, respectively.

Remark 2. One possible reading of the operations and the axioms is as follows. In **M**, we have sequential composition \circ , allocation $\langle n$ of a resource named n , and deallocation \rangle . In **G**, we don't care about deallocation (garbage collection). In **L**, the timing of the allocation does not matter and all resources may be allocated at the start. In **S**, the order of allocation does not matter and the allocation of an unused resource is redundant.

But other interpretations are possible. With $[n]$ as the νn of the π -calculus and \circ as $|$, Ax6 becomes the familiar law of scope extrusion. Interpreting $[n]$ as \forall , Ax4-5 are familiar laws of the universal quantifier. In [14], a binder satisfying Ax4-5 is called a name-restriction operator.

We can now summarise the previous section conveniently in Table 1 and

Theorem 1. **M, G, L, S** are the initial monoids in, respectively, C_m, C_g, C_l and C_s .

Table 1. Summary of nominal monoids and the axioms

| Classes | Axioms | Initial monoid | Typical example |
|---------|--------|----------------|--|
| C_m | | M | $[n_1](s_1 n_1 n_4)[n_0](n_0[n_3]s_2)$ |
| C_g | Ax1 | G | $[n_1](s_1 n_1 n_4[n_0](n_0[n_3]s_2))$ |
| C_l | Ax1-3 | L | $[n_1][n_0][n_3]s_1 n_1 n_4 n_0 s_2$ |
| C_s | Ax1-5 | S | $[n_0][n_1]s_1 n_1 n_4 n_0 s_2$ |

Proof. The detailed proof can be found in [12]. \square

Remark 3. We have a mapping $f_M : 2^{\mathbf{M}} \rightarrow 2^{(\mathcal{N} \cup \mathcal{S})^*}$ to **plain words** (ie words without binders) determined by $f_M(\{n.w\}) = f_M(\{w\}) \cup \{(n m) \cdot v \mid v \in f_M(\{w\}), m \# v\}$. With the embedding $gm \circ lg \circ sl$ from Remark 1 this induces a map f_S from languages of s -words to subsets of $(\mathcal{N} \cup \mathcal{S})^*$, eg $f_S(\{(\{n\}, n)\}) = f_M(\{n.n\}) = \mathcal{N}$.

4 Nominal Regular Expressions

In analogy to the classical definition, we introduce *nominal regular expressions*:

$$e ::= 1 \mid 0 \mid n \mid s \mid e + e \mid e \circ e \mid \langle n.e \rangle \mid e^* \quad (8)$$

where $n \in \mathcal{N}$ and $s \in \mathcal{S}$. The semantic interpretation L is defined as follows.

1. $L(1) \stackrel{\text{def}}{=} \{\epsilon\}$, $L(0) \stackrel{\text{def}}{=} \emptyset$, $L(n) \stackrel{\text{def}}{=} \{n\}$, $L(s) \stackrel{\text{def}}{=} \{s\}$,
2. $L(e_1 + e_2) \stackrel{\text{def}}{=} L(e_1) \cup L(e_2)$,
3. $L(e_1 \circ e_2) \stackrel{\text{def}}{=} L(e_1) \circ L(e_2) \stackrel{\text{def}}{=} \{w_1 \circ w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$,
4. $L(\langle n.e \rangle) \stackrel{\text{def}}{=} [n]L(e) \stackrel{\text{def}}{=} \{[n]w \mid w \in L(e)\}$.
5. $L(e^*) \stackrel{\text{def}}{=} \bigcup_{i \in \mathbb{N}} L(e)^i$, where $L(e)^i \stackrel{\text{def}}{=} \underbrace{L(e) \circ \dots \circ L(e)}_{i \text{ times}}$,

Remark 4. The definitions of \circ and $[-]$ are dependent on the choice of row in Table 1, compare (5), (6), (7). For example, on **M** we have $[n]L(e) = \{[n]w \mid w \in L(e)\}$ and on **L** we have $[n]L(e) = \{(np, w) \mid (p, w) \in L(e)\}$. From § 5 onwards, we will interpret regular expressions in **M** only.

Example 1. We have seen in (4) how $\langle n.n \langle m.nm \rangle \rangle^*$ arises from the Needham-Schroeder protocol. In § 6 we consider the simpler expression $m(\langle n.mn \rangle)^*$ which intuitively represent the computations of a security protocol where (an unbound number of) new ‘nonces’ n are generated within a session m and should always be paired up with m . \diamond

We can also interpret nominal regular expressions in plain words. For example, let L take values in **S** and let $f_S : 2^{\mathbf{S}} \rightarrow 2^{(\mathcal{N} \cup \mathcal{S})^*}$ denote the map of Remark 3.

Example 2. If we interpret $\langle n.n \rangle^*$ in **S** (or **L** or **G**) we obtain the language

$$\mathcal{L}_2 \stackrel{\text{def}}{=} f_S(L(\langle n.n \rangle^*)) = \{n_1 \cdots n_k \mid \forall i \neq j. n_i \neq n_j\},$$

which is the complement of \mathcal{L}_1 from (1). \mathcal{L}_2 is not recognised by the FMAs of [11] but it is recognised by the FRAs of [17]. The latter notes that $\mathcal{L}_2 * \mathcal{L}_2 = \{wv \mid w, v \in \mathcal{L}_2\}$ shows that languages recognised by FRAs are not closed under composition. On the other hand, the presence of binders allows us to use \circ (respecting the ‘hidden’ binders) instead of $*$ and we obtain $\mathcal{L}_2 \circ \mathcal{L}_2 = f_S(L(\langle n.n \rangle^*)) \circ f_S(L(\langle n.n \rangle^*)) \stackrel{\text{def}}{=} f_S(L(\langle n.n \rangle^* \circ \langle n.n \rangle^*)) = f_S(L(\langle n.n \rangle^*)) = \mathcal{L}_2$, where the second equality is our definition of ‘nominal concatenation’ on languages of plain words. This indicates that even for languages without binders the composition with binders is a natural concept.

Similarly, if we interpret $e = \langle l.l \circ \langle m.\langle n.m \circ n \rangle^* \rangle^* \rangle$ in **M** we obtain another example of Tzevelekos:

$$f_M(L(e)) = \{mn_1^n n_2^n \cdots n_k^n n_k^n \mid \forall i \in \mathbb{N}, \forall j \in \{1, 2\}. m \neq n_i^j \& n_i^1 \neq n_i^2\}$$

5 History-dependent Automata with Stack

We build our nominal automata theory on HDA (after *history-dependent automata*) [13]. HDA are a computational model of nominal calculi defined on the notion of *named sets* and extend classical automata with finite sets of names *local* to states and transitions. We equip HDA with stack; this renders them suitable for recognising nominal languages interpreted in **M**. We argue that HDA are natural candidates to build a theory of automata of nominal languages with binders. In fact, they are equipped with mechanisms to capture name restriction of nominal calculi [3, 5, 4] and formally linked to the nominal set theory in [10, 6].

Let $\star \notin \mathcal{N}$ be a distinguished name; a *stack* Σ is a sequence of finite partial maps $\sigma : \mathcal{N} \rightarrow \mathcal{N} \cup \{\star\}$ (we use \perp to denote the empty map). The empty stack is denoted by \emptyset , a stack with head σ is written $\sigma :: \Sigma$, and

$$\hat{\Sigma} \stackrel{\text{def}}{=} \begin{cases} \Sigma', & \Sigma = \sigma :: \Sigma' \\ \emptyset, & \Sigma = \emptyset \end{cases} \quad \hat{\hat{\Sigma}} \stackrel{\text{def}}{=} (\hat{\Sigma}) \quad \Sigma^\top \stackrel{\text{def}}{=} \begin{cases} \sigma, & \Sigma = \sigma :: \Sigma' \\ \perp, & \Sigma = \emptyset \end{cases}$$

respectively are the pop, pop twice, and top operations.

Definition 8 ([13]). A (basic) named set $\langle Q, |-|_Q \rangle$ is a set Q (of states) with a map $|-|_Q : Q \rightarrow \mathcal{P}_\omega \mathcal{N}$ sending $q \in Q$ to a finite set of names $|q|_Q$ (called local names of q).

Basically, the elements q of a named set are equipped with a set of *local* names $|q|_Q$. Hereafter we omit subscripts when clear from the context and write a named set $\langle Q, |-|_Q \rangle$ as Q , in which case $|-|$ is understood as the map of local names of Q ; also, the *update* of a map $f : X \rightarrow Y$ at x with y is the map

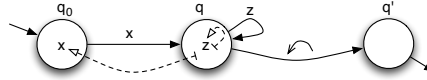
$$f[x \mapsto y] : X \cup \{x\} \rightarrow Y \cup \{y\} \quad \text{such that} \quad (f[x \mapsto y])(a) = \begin{cases} f(a), & \text{if } a \neq x \\ y, & \text{if } a = x \end{cases}$$

Before giving the formal definition, we intuitively present HDA with stack. A transition $\langle q', \alpha, \sigma \rangle$ from a state q consists of the target state q' , a label α , and a map σ keeping track of the correspondences of names. Labels α can be a local name $n \in |q|$ of the source state q , letters $s \in \mathcal{S}$, or any of the distinguished symbols

$$\varepsilon \quad \curvearrowright \quad \curvearrowleft \quad \langle \quad \rangle$$

respectively representing internal transitions, push, pop, *name allocation*, and *name deallocation*. Example 3 gives a convenient graphical representation of an HDS.

Example 3. Let q_0 , q , and q' be states with $|q_0| = \{x\}$, $|q| = \{z\}$, and $q' = \emptyset$. The HDS



has initial (resp. final) state q_0 (resp. q'). Both q_0 and q have a transition exposing their (unique) local name (x and z respectively). Maps among local names are represented by dashed arrows. Also, q has \curvearrowleft transition to q' with the empty map of local names. \diamond

Definition 9. A (non-deterministic) history-dependent automaton with stack on $\mathcal{N} \cup \mathcal{S}$ (HDS) is a tuple $\langle Q, q_0, \eta, F, tr \rangle$ where

- Q is a named set of states (the states of the automaton);
- $q_0 \in Q$ is the initial state;
- η is a partial function from $|q_0|_Q$ to \mathcal{N} ;
- $F \subseteq Q$ is the named set of final states with $|-|_F$ being the restriction of $|-|_Q$ to F ;
- tr is the transition function returning for each $q \in Q$ a finite set $tr(q)$ of transitions, namely tuples $\langle q', \alpha, \sigma \rangle$ such that
 - if $\alpha \in \mathcal{N}$ then $\alpha \in |q|_Q$
 - if $\alpha = \langle \rangle$ then $\sigma : |q'| \rightarrow |q| \cup \{\star\}$
 - if $\alpha = \curvearrowright$ then $\sigma : |q'| \rightarrow \mathcal{N}$
 - otherwise $\sigma : |q'| \rightarrow |q|$

and, in either case, σ is a partial injective map (see Remark 6 on page 11).

Transitions in Def 9 allow HDS to accept names or letters or to manipulate the stack. Besides the usual *push* (\curvearrowright) and *pop* (\curvearrowleft) operations, HDS feature allocation ($\langle \rangle$) and deallocation (\rangle) of names.

Example 4. Let $Q = \{q_0, q, q'\}$, $F = \{q'\}$. The HDS $H = \langle Q, q_0, \eta, F, tr \rangle$ where

$$|-| : \begin{cases} q_0 \mapsto \{x\} \\ q \mapsto \{z\} \\ q' \mapsto \emptyset \end{cases} \quad tr : \begin{cases} q_0 \mapsto \{\langle q, x, \sigma : z \rightarrow x \rangle\} \\ q \mapsto \{\langle q', \curvearrowright, \perp \rangle, \langle q, z, \sigma_1 : z \rightarrow z \rangle\} \\ q' \mapsto \emptyset \end{cases}$$

formally defines the HDS in Example 3 (where η is not represented for simplicity). \diamond

We now define how HDS can recognise languages of \mathbf{M} . Hereafter, we fix an HDS

$$\mathcal{H} \stackrel{\text{def}}{=} \langle Q, q_0, \eta, F, tr \rangle \quad (9)$$

and, for any stack Σ and any name mapping σ , we define $\Sigma \bullet \sigma$ by

$$\emptyset \bullet \sigma \stackrel{\text{def}}{=} \sigma :: \emptyset \quad \text{and} \quad \Sigma \bullet \sigma \stackrel{\text{def}}{=} (\Sigma^\top)[\star \mapsto \star] \circ \sigma :: (\widehat{\Sigma})$$

that basically updates Σ by post-composing its top map Σ^\top (if any) with σ . Note that this requires Σ^\top to be updated to allow composition when $\star \in \text{cod}(\sigma)$.

A *configuration* of \mathcal{H} in (9) is a triple $\langle q, w, \Sigma \rangle$ where $q \in Q$, w is an \mathbf{M} , and Σ is a stack. Call *initial* a configuration $\langle q_0, w, \eta :: \emptyset \rangle$ and *accepting* $\langle q, \varepsilon, \Sigma \rangle$ if $q \in F$.

Definition 10. Given $q, q' \in Q$ and two configurations $t = \langle q, w, \Sigma \rangle$ and $t' = \langle q', w', \Sigma' \rangle$, \mathcal{H} in (9) moves from t to t' (written $t \xrightarrow{\mathcal{H}} t'$) iff there is $\langle q', \alpha, \sigma \rangle \in \text{tr}(q)$ such that either of the following cases applies

$$\left\{ \begin{array}{ll} \alpha \in |q| & \implies w = nw' \wedge \Sigma^\top(\alpha) = n \wedge \Sigma' = \Sigma \bullet \sigma \\ \alpha = s \in \mathcal{S} & \implies w = sw' \wedge \Sigma' = \Sigma \bullet \sigma \\ \alpha = \varepsilon & \implies w' = w \wedge \Sigma' = \Sigma \bullet \sigma \\ \alpha = \curvearrowright & \implies w' = w \wedge \Sigma' = \sigma :: \Sigma \\ \alpha = \curvearrowleft & \implies w' = w \wedge \Sigma' = \sigma' :: \widehat{\Sigma}^{\curvearrowright} \text{ where } \sigma' = \widehat{\Sigma}^\top \circ \sigma \\ \alpha = \llbracket & \implies w = \llbracket n.w' \wedge \Sigma' = \sigma' :: \Sigma, \text{ where } \sigma' = (\Sigma^\top[\star \mapsto n]) \circ \sigma \\ \alpha = \rrbracket & \implies w = \rrbracket w' \wedge \Sigma' = \sigma' :: \widehat{\Sigma}^{\curvearrowright}, \text{ where } \sigma' = \widehat{\Sigma}^\top \circ \sigma \end{array} \right.$$

The set $\text{rec}_{\mathcal{H}}(t)$ of states reached by \mathcal{H} from t on w is defined as

$$\text{rec}_{\mathcal{H}}(t) \stackrel{\text{def}}{=} \begin{cases} \{q\} & \text{if } t = \langle q, \varepsilon, \Sigma \rangle \\ \bigcup_{t \xrightarrow{\mathcal{H}} t'} \text{rec}_{\mathcal{H}}(t') & \text{otherwise} \end{cases}$$

A run of \mathcal{H} on an m -word w is a sequence of moves of \mathcal{H} from $\langle q_0, w, \eta :: \emptyset \rangle$.

Intuitively, HDS “consume” the word in input *moving* from one configuration to another (likewise classical automata). However, when the current word starts with a name n , the automaton can progress only if the name “is known”; namely, it is necessary to find a transition from the current state q for which the stack maps a local name of q to n .

HDS use a stack (i) to keep track of the names of the current state and, noticeably, (ii) to (de)allocate bound names in input strings. More precisely, a binder is consumed using a \llbracket transition which updates the meaning of the names. This is basically done by post-composing the mapping σ in the selected transition with the map on the top of the stack (opportune updated to take into account the allocation of n). Instead, a \rrbracket transition will pop the stack so reassigning previous meanings to names in the current state by post-composing the map σ of the transition with “the second one” in the stack.

An automaton \mathcal{H} recognises w if it has a run from its initial state to a final state that consumes w .

Definition 11. The HDS \mathcal{H} in (9) accepts (or recognises) w if $F \cap \text{rec}_{\mathcal{H}}(\langle q_0, w, \eta :: \emptyset \rangle) \neq \emptyset$. The language of \mathcal{H} (written $\mathcal{L}_{\mathcal{H}}$) is the set of words accepted by \mathcal{H} .

Example 5. If \mathcal{H} is the HDS in Example 4 and $\eta : x \mapsto n$, then $\mathcal{L}_{\mathcal{H}} = \{n^i \mid i > 0\}$. \diamond

Defs 10 and 11 contain some subtleties worth spelling out. First, observe that the language recognised by \mathcal{H} depends on η which intuitively sets the meaning of the local names of the initial state q_0 ; instead, the language of \mathcal{H} does not depend on the identities of the local names of the states in \mathcal{H} . Secondly, an alternative definition would allow the initial stack to be empty and the correspondence between local names of the states of \mathcal{H} and those in the input word is incrementally built during recognition. This class of HDSs would be equivalent to the one in Defs 9 and 10, but it would have made our constructions more complex. Finally, as for classical push-down automata, we could have equivalently required that an HDS recognises an m -word w only when it has a run leading to a final state that consumes w and empties the stack. We opted for Def 11 as it is conceptually simpler. For instance, the following lemma (used to prove Proposition 3) states that only the top of the stack is relevant for accepting words.

Lemma 1. Any configuration reachable by an HDS as in (9) from $\langle q_0, w, \eta :: \emptyset \rangle$ is also reachable from $\langle q_0, w, \eta :: \Sigma \rangle$ for any stack Σ . \square

In § 6 we show how a nominal regular expression e can be mapped on an HDS $\langle e \rangle$ that recognises the language of e . Theorem 2 is the main result

Theorem 2. For each nominal regular expression e , $\mathcal{L}_{\langle e \rangle} = L(e)$ interpreted on \mathbf{M} .

Proof. The proof is by induction on the structure of e . The base cases are trivial while the other cases follow by Propositions 1, 2, 3, and 4. \square

6 HDS and Nominal Regular Expressions

We use nominal regular expressions (8) to establish a correspondence between HDS and nominal formal languages. More precisely, we give (Def 12) the map mentioned in Theorem 2 as the homomorphic image of nominal regular expression on an algebra of HDS given in the rest of this section.

Definition 12. The map $\langle _ \rangle$ from nominal regular expressions to HDS is defined as:

$$\begin{aligned} \langle 1 \rangle &= \langle \{q_0, q\}, q_0, \perp, \{q\}, q_0 \mapsto \{ \langle q, \varepsilon, \perp \rangle \} \rangle \text{ where } |q_0| = |q| = \emptyset \\ \langle 0 \rangle &= \langle \{q_0\}, q_0, \perp, \emptyset, q_0 \mapsto \emptyset \rangle \text{ where } |q_0| = \emptyset \\ \langle n \rangle &= \langle \{q_0, q\}, q_0, x \mapsto n, \{q\}, q_0 \mapsto \{ \langle q, x, \perp \rangle \} \rangle \text{ where } |q_0| = \{x\}, |q| = \emptyset \\ \langle s \rangle &= \langle \{q_0, q\}, q_0, \perp, \{q\}, q_0 \mapsto \{ \langle q, s, \perp \rangle \} \rangle \text{ where } |q_0| = |q| = \emptyset \\ \langle e_1 + e_2 \rangle &= \langle e_1 \rangle + \langle e_2 \rangle \\ \langle e_1 \circ e_2 \rangle &= \langle e_1 \rangle \circ \langle e_2 \rangle \\ \langle e^* \rangle &= \langle e \rangle^* \\ \langle \langle n.e \rangle \rangle &= [n] \langle e \rangle \end{aligned}$$

where the operations on HDS in the last four cases are defined in the following.

The operations on HDS in Def 12 allow to combine them so that the language of the resulting HDS has a clear relation with those the operations act upon as per Propositions 1, 2, 3, and 4 below. Theorem 2 can be proved by induction on the structure of nominal regular expressions using such propositions.

Remark 5. The map $(\lfloor _ \rfloor)$ in Def 12 depends on the choice of local names; however, as noted in § 5, recognisability does not depend on the identity of such names.

The first two clauses in Def 6 do not involve names and stack. Notably, the third clause states that the HDS corresponding to an expression n has simply a transition from the initial to accepting state and in the initial configuration the unique name of the former is mapped to n .

The set $|\mathcal{H}|$ of (local) names of an HDS \mathcal{H} as in (9) is defined as $|\mathcal{H}| \stackrel{\text{def}}{=} \bigcup_{q \in Q} |q|$. In the following, we fix two HDS

$$\mathcal{H}_i \stackrel{\text{def}}{=} \langle Q_i, q_{0,i}, \eta_i, F_i, tr_i \rangle \text{ for } i \in \{1, 2\} \quad (10)$$

and, without loss of generality, we assume that $Q_1 \cap Q_2 = \emptyset$ and $|\mathcal{H}_1| \cap |\mathcal{H}_2| = \emptyset$.

Definition 13. Let $q_0 \notin Q_1 \cup Q_2$ be a new state. We define $\mathcal{H}_1 + \mathcal{H}_2$ to be the automaton $\mathcal{H}^+ = \langle Q^+, q_0^+, \eta^+, F^+, tr^+ \rangle$ where

- $Q^+ = Q_1 \cup Q_2 \cup \{q_0^+\}$ where $|q_0^+|_{Q^+} = |q_{0,1}|_{Q_1} \cup |q_{0,2}|_{Q_2}$ and $F^+ = F_1 \cup F_2$
- $tr^+(q_0^+) = \{ \langle q_{0,i}, \varepsilon, id_{|q_{0,i}|_{Q_i}} \rangle \mid \text{for } i \in \{1, 2\} \}$ and $tr^+|_{Q_i} = tr_i$ for $i \in \{1, 2\}$, where $id_{|q_{0,i}|_{Q_i}}$ is the identity from $|q_{0,i}|_{Q_i}$ to $|q_0^+|_{Q^+}$
- $\eta^+ = \eta_1 + \eta_2$, namely $\eta^+(x) = \eta_i(x)$ if $x \in |q_{0,i}|_{Q_i}$.

Proposition 1. $\mathcal{L}_{\mathcal{H}^+} = \mathcal{L}_{\mathcal{H}_1} \cup \mathcal{L}_{\mathcal{H}_2}$

Proof. The statement trivially follows from Def 10 as (i) q_0 has only two outgoing ε -transitions which lead to the initial states of either of \mathcal{H}_1 or \mathcal{H}_2 and (ii) η preserves the name assignments η_1 and η_2 . \square

Lemma 2. For each HDS \mathcal{H} there is an HDS \mathcal{H}' with a unique final states and such that $\mathcal{L}_{\mathcal{H}} = \mathcal{L}_{\mathcal{H}'}$.

Proof. Given \mathcal{H} in (9) and $\hat{q} \notin Q$ such that $|\hat{q}| = \emptyset$, we define $\mathcal{H}' \stackrel{\text{def}}{=} \langle Q \cup \{\hat{q}\}, q_0, \eta, \{\hat{q}\}, tr' \rangle$ where $tr'(\hat{q}) = \emptyset$, $tr' = tr$ when restricted to $Q \setminus F$, and $tr'(q) = tr(q) \cup \{ \langle \varepsilon, q \rangle \perp \}$ for each $q \in F$. The proof that $\mathcal{L}_{\mathcal{H}} = \mathcal{L}_{\mathcal{H}'}$ is similar to the proof of Proposition 1. \square

Lemma 2 allows, without loss of generality, \mathcal{H} in (9) and each of \mathcal{H}_1 and \mathcal{H}_2 in (10) to have a single final state, namely $F = \{q_f\}$, $F_1 = \{q_{f,1}\}$ and $F_2 = \{q_{f,2}\}$, respectively.

The following construction extends the names of an HDS without altering its language and is used in Def 15.

Definition 14. Given \mathcal{H} as in (9) and $x \in \mathcal{N} \setminus |\mathcal{H}|$, $\mathcal{H}^\dagger x = \langle Q^\dagger, q_0^\dagger, \eta^\dagger, F^\dagger, tr^\dagger \rangle$ is the HDS such that

- Q^\dagger is the named set having the same elements of Q with $|-|_{Q^\dagger} : q \mapsto |q|_Q \cup \{x\}$

- F^\dagger is the named set with the same states of F and $|-|_{F^\dagger} : q \mapsto |q|_{Q^\dagger} \cup \{x\}$
- $tr^\dagger(q) = \{(q', \alpha, \sigma[x \mapsto x]) \mid (q', \alpha, \sigma) \in tr(q)\}$
- $\eta^\dagger : |q|_{Q^\dagger} \cup \{x\} \rightarrow \mathcal{N}$ is the partial map undefined on x and behaving as η otherwise.

Hereafter, we assume that $x \in \mathcal{N} \setminus |\mathcal{H}|$ when writing $\mathcal{H}^\dagger x$; in fact, by the locality of the names in the states of an HDS, if q is a state of \mathcal{H} such that $x \in |q|$, we can replace x with any name not in $|q|$ by rearranging all the maps in the transitions reaching q .

Lemma 3. $\mathcal{L}_{\mathcal{H}^\dagger x} = \mathcal{L}_{\mathcal{H}}$.

Proof. The proof that $\mathcal{L}_{\mathcal{H}} \subseteq \mathcal{L}_{\mathcal{H}^\dagger x}$ is trivial as all the transitions of \mathcal{H} have a correspondent in $\mathcal{H}^\dagger x$ with exactly the same labels and name mappings. The converse also hold trivially as x cannot play any role in the recognition of a word in $\mathcal{H}^\dagger x$ as η' is not defined on x . \square

Definition 15. Let $\{x_1, \dots, x_j\} = |q_{0,2}|$ and $(\dots(\mathcal{H}_1^\dagger x_1) \dots \dagger)x_j = \langle Q'_1, q'_{0,1}, \eta', \{q'_{f,1}\}, tr' \rangle$. The HDS $\mathcal{H}_1 \circ \mathcal{H}_2$ is defined as $\langle Q^\circ, q'_{0,1}, \eta^\circ, F_2, tr^\circ \rangle$ where $Q^\circ = Q'_1 \cup Q_2$ and

$$\eta^\circ(x) = \begin{cases} \eta_2(x) & x \in |q_{0,2}| \\ \eta'(x) & \text{otherwise} \end{cases} \quad tr^\circ(q) = \begin{cases} tr'(q) & q \in Q'_1 \setminus \{q'_{f,1}\} \\ tr_2(q), & q \in Q_2 \\ tr'(q) \cup \{\langle q_{0,2}, \varepsilon, id_{|q_{0,2}|} \rangle\}, & q = q'_{f,1} \end{cases}$$

The HDS $\mathcal{H}_1 \circ \mathcal{H}_2$ is built by connecting the accepting state of \mathcal{H}_1 to $q_{0,2}$, the initial state of \mathcal{H}_2 , after adding $|q_{0,2}|$ to \mathcal{H}_1 . Note that the newly introduced ε -transition maintains the initial meaning of the names in $|q_{0,2}|$ since η° acts as η' on $|q_{0,2}|$ (and by Def 14).

Remark 6. A definition more complex than Def 15 can be given to preserve the injectivity of the initial mapping η° when η_1 and η_2 are injective. This requires to relax the injectivity condition on σ in Def 9 requiring $\sigma(x) = \sigma(y) \iff \sigma(x) = \star$ for any $x, y \in \text{dom}(\sigma)$. We opted for the simpler Def 15 as it just allows more non-determinism without altering the expressiveness of HDS.

Proposition 2. $\mathcal{L}_{\mathcal{H}_1 \circ \mathcal{H}_2} = \mathcal{L}_{\mathcal{H}_1} \circ \mathcal{L}_{\mathcal{H}_2}$.

Proof. The automaton $\mathcal{H}_1 \circ \mathcal{H}_2$ reaches a final state iff $w = w_1 w_2$ where $w_i \in \mathcal{L}_{\mathcal{H}_i}$ for $i = 1, 2$. In fact, to reach $q_{f,2}$ it is necessary to reach $q_{f,1}$ first and the unique transition from $q_{f,1}$ to $q_{0,2}$ maintains on the stack the meaning assigned to the names $|q_{0,2}|$ as per the stack. \square

Definition 16. Let \mathcal{H} be as in (9) with $F = \{q_f\}$. The HDS $\mathcal{H}^* = \langle Q, q_0, \eta, \{q_f\}, tr^* \rangle$ is such that

$$\begin{aligned} tr^*(q) &= tr(q), \quad \text{for all } q \in Q \setminus \{q_0, q_f\} \\ tr^*(q_0) &= tr(q_0) \cup \{\langle q_f, \varepsilon, \perp \rangle\} \\ tr^*(q_f) &= \{\langle q_0, \curvearrowright, \eta \rangle\} \end{aligned}$$

The construction of \mathcal{H}^* simply adds an ε -transition from q_0 (the initial state of \mathcal{H}) to q_f (the accepting state of \mathcal{H}) and a \curvearrowright -transition from q_f to q_0 that re-establish the mapping of the initial configuration preserving in the stack the meaning of the names.

Proposition 3. $\mathcal{L}_{\mathcal{H}^*} = \mathcal{L}_{\mathcal{H}}^*$

Proof. (Sketch.) First, observe that trivially $\varepsilon \in \mathcal{L}_{\mathcal{H}^*} \cap \mathcal{L}_{\mathcal{H}}^*$ because \mathcal{H}^* has a transition $\langle \hat{q}, \varepsilon, \perp \rangle$ from q_0 .

We now prove that $\mathcal{L}_{\mathcal{H}^*} \subseteq \mathcal{L}_{\mathcal{H}}^*$. If $w \neq \varepsilon \in \mathcal{L}_{\mathcal{H}^*}$ then \mathcal{H}^* reaches a configuration $\langle \hat{q}, \varepsilon, \Sigma \rangle$ for a suitable Σ . By construction and Def 10, \mathcal{H}^* can visit \hat{q} only a finite number of times k . Hence, $w = w_1 \circ \dots \circ w_k$ where w_i is the word processed between the i -th visit of \hat{q} and the previous visit of \hat{q} (or of q_0 if $i = 1$).

Observing that each visit of \hat{q} is preceded by a visit of q_f (since \hat{q} can only be reached through q_f), we have that $w_1 \in \mathcal{L}_{\mathcal{H}}$ (and hence in $\mathcal{L}_{\mathcal{H}}^*$) because there q_f can be reached from the configuration $\langle q_0, w_1, \eta :: \emptyset \rangle$. For the same reason, we can conclude that $w_{i+1} \in \mathcal{L}_{\mathcal{H}}$ for each $i \in \{1, \dots, k-1\}$; in fact, the i -th visit of \hat{q} yields \mathcal{H}^* in the configuration $\langle \hat{q}, w_{i+1} \circ \dots \circ w_k, \Sigma \rangle$ for some stack Σ . Hence, using the unique transition $\langle q_0, \circlearrowleft, \eta \rangle$ from \hat{q} , the automaton “resets” to the configuration $\langle q_0, w_{i+1} \circ \dots \circ w_k, \eta :: \Sigma \rangle$, which basically amounts to say that w_i can be recognised by \mathcal{H} and the next work w_{i+1} is processed from a configuration where η is on the top of the stack and the thesis follows by Lemma 1.

We prove that $\mathcal{L}_{\mathcal{H}}^* \subseteq \mathcal{L}_{\mathcal{H}^*}$. Any word $w \in \mathcal{L}_{\mathcal{H}}^*$ has the form $w = w_1 \circ \dots \circ w_k$ where $w_i \in \mathcal{L}_{\mathcal{H}}$ for each $i \in \{1, \dots, k\}$, so we proceed by induction on k . If $k = 0$ the thesis follows trivially. If $k > 0$ then, from the configuration $\langle q_0, w_1 \circ w_2 \circ \dots \circ w_k, \eta :: \emptyset \rangle$, \mathcal{H}^* reaches a configuration $\langle q_f, w_2 \circ \dots \circ w_k, \Sigma \rangle$ since $w_1 \in \mathcal{L}_{\mathcal{H}}$ by hypothesis. Since $\langle \hat{q}, \varepsilon, \perp \rangle \in \text{tr}'(q_f)$, the configuration $\langle \hat{q}, w_2 \circ \dots \circ w_k, \Sigma \bullet \perp \rangle$ is reachable from \mathcal{H}^* . Therefore, \mathcal{H}^* reaches the configuration $\langle q_0, w_2 \circ \dots \circ w_k, \eta :: \Sigma \bullet \perp \rangle$ which yields the thesis by Lemma 1. \square

Definition 17. Let $n \in \mathcal{N}$, \mathcal{H} be as in (9) with $F = \{q_f\}$, and let $\hat{q}, \hat{q}_f \notin Q$ be new states with $|\hat{q}| = |q_0| \setminus \eta^{-1}(n)$ and $|\hat{q}_f| = \emptyset$. The HDS $[n]\mathcal{H} = \langle Q \cup \{\hat{q}, \hat{q}_f\}, \hat{q}, \eta|_{|\hat{q}|}, \{\hat{q}_f\}, \text{tr}' \rangle$ is such that

$$\text{tr}'(\hat{q}) = \{\langle q_0, \circlearrowleft, \sigma \rangle\}, \quad \text{tr}'(q) = \text{tr}(q), \quad \forall q \in Q \setminus \{q_f\}, \quad \text{tr}'(q_f) = \text{tr}(q_f) \cup \{\langle \hat{q}_f, \circlearrowright, \perp \rangle\}$$

where $\sigma = \text{id}_{|q_0|}[x \mapsto \star]$, if $\eta^{-1}(n) = \{x\}$, otherwise $\sigma = \text{id}_{|q_0|}$.

Proposition 4. $\mathcal{L}_{[n]\mathcal{H}} = [n]\mathcal{L}_{\mathcal{H}}$.

Proof. By construction, $\langle \hat{q}, w, \eta|_{|\hat{q}|} :: \emptyset \rangle$ reaches \hat{q}_f iff there is a word w' such that $w = \circlearrowleft n.w'$ and $\langle q_0, w', \sigma' \rangle$ reaches \hat{q}_f where σ' is built as in Def 10. Again by construction, this is possible iff $\langle q_0, w', \sigma' \rangle$ visits q_f and the last transition which consumes the word is a (deallocation) \circlearrowright -transition from q_f to \hat{q}_f . This is equivalent to say that there is $w'' \in \mathcal{L}_{\mathcal{H}}$ such that $w' = w''$ which, by Remark 4, yields the thesis. \square

7 Mapping Nominal Regular Expressions to HDS

We build the HDS $\mathcal{H}_{[m](\langle n.mn \rangle)^*}$ corresponding to the expression $m(\langle n.mn \rangle)^*$ by applying the constructions of § 6. By Definition 12, the HDS corresponding to the expression m is $\mathcal{H}_{[m]}$ with

$$[m] = \mathcal{H}_{[m]} = \langle \{q_{0,m}, q_{f,m}\}, q_{0,m}, \eta_m, \{q_{f,m}\}, \text{tr}_m \rangle \quad (11)$$

where $|q_{0,m}| = \{x\}$, $|q_{f,m}| = \emptyset$, $\eta_m: x \mapsto m$, $tr_m: q_{f,m} \mapsto \emptyset$, and $tr_m: q_{0,m} \mapsto \{\langle q_{f,m}, x, \perp \rangle\}$. Analogously, the HDS corresponding to the expression n is $\mathcal{H}_{\langle n \rangle}$ with

$$\langle n \rangle = \mathcal{H}_{\langle n \rangle} = \langle \{q_{0,n}, q_{f,n}\}, q_{0,n}, \eta_n, \{q_{0,n}\}, tr_n \rangle$$

where $|q_{0,n}| = \{y\}$, $|q_{f,n}| = \emptyset$, $\eta_n: y \mapsto n$, $tr_n: q_{f,n} \mapsto \emptyset$, and $tr_n: q_{0,n} \mapsto \{\langle q_{f,n}, \perp, y \rangle\}$.

To compose $\mathcal{H}_{\langle m \rangle}$ and $\mathcal{H}_{\langle n \rangle}$, we first have to compute $\mathcal{H}_{\langle m \rangle} \dagger y$; by Def 14, $\mathcal{H}_{\langle m \rangle} \dagger y = \langle Q_{\dagger}, q_{0,\dagger}, \eta_{\dagger}, \{q_{f,\dagger}\}, tr_{\dagger} \rangle$ where $Q_{\dagger} = \{q_{0,\dagger}, q_{f,\dagger}\}$ and

$$|-|_{\dagger}: \begin{cases} q_{0,\dagger} \mapsto \{x, y\} \\ q_{f,\dagger} \mapsto \{y\} \end{cases} \quad \eta_{\dagger}: \begin{cases} x \mapsto m \\ y \mapsto \perp \end{cases} \quad tr_{\dagger}: \begin{cases} q_{0,\dagger} \mapsto \{\langle q_{f,m}, x, y \mapsto y \rangle\} \\ q_{f,\dagger} \mapsto \emptyset \end{cases}$$

By Def 15, $\mathcal{H}_{\langle m \rangle} \circ \mathcal{H}_{\langle n \rangle} = \langle Q_{\circ}, q_{0,\circ}, \eta_{\circ}, \{q_{f,n}\}, tr_{\circ} \rangle$ where $Q_{\circ} = \{q_{0,\dagger}, q_{f,\dagger}, q_{0,n}, q_{f,n}\}$,

$$\eta_{\circ}: \begin{cases} x \mapsto m \\ y \mapsto n \end{cases} \quad \text{and} \quad tr_{\circ}: \begin{cases} q_{0,\dagger} \mapsto \{\langle q_{f,\dagger}, x, y \mapsto y \rangle\} \\ q_{f,\dagger} \mapsto \{\langle q_{0,n}, \varepsilon, y \mapsto y \rangle\} \\ q_{0,n} \mapsto \{\langle q_{f,n}, y, \perp \rangle\} \\ q_{f,n} \mapsto \emptyset \end{cases}$$

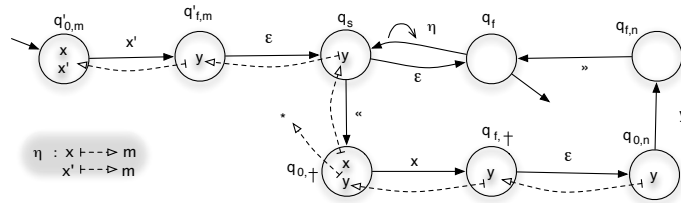
We now build $\mathcal{H}_{\langle (n.mn) \rangle} = [n](\mathcal{H}_{\langle m \rangle} \circ \mathcal{H}_{\langle n \rangle})$; let q_s and q_t be two new states with $|q_s| = \{x\}$ and $|q_t| = \emptyset$, as prescribed by Def 17, we have $\mathcal{H}_{\langle (n.mn) \rangle} = \langle Q_{[n]}, q_s, \eta_{[n]}, \{q_t\}, tr_{[n]} \rangle$ where $Q_{[n]} = Q_{\circ} \cup \{q_s, q_t\}$ and the initial setting $\eta_{[n]}$ by restricting η_{\circ} on $|q_s|$, i.e. $\eta_{[n]}: x \mapsto m$; moreover,

$$tr_{[n]}: \begin{cases} q_s \mapsto \{\langle q_{0,\dagger}, \varepsilon, \sigma \rangle\}, & \text{where } dom(\sigma) = \{x, y\} \text{ and } \sigma(x) = y \text{ and } \sigma(y) = \star \\ q_{f,n} \mapsto \{\langle q_t, \varepsilon, \perp \rangle\} \\ q \mapsto tr_{\dagger}(q), & \text{if } q \in Q_{\dagger} \setminus \{q_{f,n}\} \end{cases}$$

Further, by Def 16, $\mathcal{H}_{\langle (n.mn) \rangle}^*$ is obtained by adding two extra transitions $q_s \mapsto \{\langle q_t, \varepsilon, \perp \rangle\}$ and $q_t \mapsto \{\langle q_s, \varepsilon, \eta_{[n]} \rangle\}$.

Finally, by Def 15, we obtain the HDS $\mathcal{H}_{\langle m \rangle \langle (n.mn) \rangle}^*$ as follows. First, let $\mathcal{H}_{\langle m \rangle}^* = \langle \{q'_{0,m}, q'_{f,m}\}, q'_{0,m}, \eta'_m, \{q'_{f,m}\}, tr'_m \rangle$ be obtained as in (11) by defining $|q'_{0,m}| = \{x'\}$, $|q'_{f,m}| = \emptyset$, $\eta'_m: x' \mapsto m$, $tr'_m: q'_{f,m} \mapsto \emptyset$, and $tr'_m: q'_{0,m} \mapsto \{\langle q'_{f,m}, x', \perp \rangle\}$. Then, we set $\mathcal{H}_{\langle m \rangle \langle (n.mn) \rangle}^* = \mathcal{H}_{\langle m \rangle}^* \circ \mathcal{H}_{\langle (n.mn) \rangle}^* = \langle Q, q'_{0,m}, \eta, \{q_t\}, tr \rangle$ where $Q = Q_{[n]} \cup \{q'_{0,m}, q'_{f,m}\}$ and

$$\eta: \begin{cases} x' \mapsto m \\ x \mapsto m \end{cases} \quad tr: \begin{cases} q \mapsto tr_{[n]}(q), & \text{if } q \in Q_{[n]} \\ q'_{0,m} \mapsto tr'_m(q'_{f,m}) \cup \{\langle q'_{f,m}, \varepsilon, \perp \rangle\} \\ q'_{f,m} \mapsto tr'_m(q'_{f,m}) \cup \{\langle q_s, \varepsilon, y \mapsto y \rangle\} \end{cases}$$



We conclude with some final remarks. Equivalent definitions could have been adopted; for instance, η above is not required to be injective (adding some non-determinism in Def 10) or some of the new states introduced by the constructions above could be avoided to obtain more compact HDS. We decided to use conceptually simpler constructions instead of more effective, but more complex ones.

8 Conclusion

This paper developed the beginnings of a general theory of words with binders: nominal languages, nominal monoids, nominal regular expressions, HD-automata with stacks. We sketch some further work.

Coming back to Table 1 further classes maybe relevant, for example words satisfying Ax4-5 but not Ax1-3; it will also be of interest to mix different binders each obeying its own axioms plus further axioms of their interaction.

HD-automata with stacks are more powerful than necessary if one is only interested in recognising regular languages; a restricted class of HD-automata characterising regular languages of m-words can be described; the same should be done for g-words, l-words, and s-words.

We will also investigate the connections (cf. Example 2) of our nominal languages with languages (on infinite alphabets) without binders [1, 17, 7, 2].

Further, closure properties and decidability results for these classes of automata should be studied; for verification purposes deterministic and minimal automata will be of interest.

Last but not least, case studies showing the relevance of this line of research to verification will have to be explored.

References

1. M. Bojanczyk. Data monoids. In *STACS'11*.
2. V. Ciancia and E. Tuosto. A novel class of automata for languages on infinite alphabets. Technical Report CS-09-003, Leicester, 2009.
3. G. Ferrari, U. Montanari, and E. Tuosto. Model Checking for Nominal Calculi. In *FoSSaCS'05*.
4. G. Ferrari, U. Montanari, and E. Tuosto. Coalgebraic Minimisation of HD-automata for the π -Calculus in a Polymorphic λ -Calculus. *TCS*, 331, 2005.
5. G. Ferrari, U. Montanari, E. Tuosto, B. Victor, and K. Yemane. Modelling and Minimising the Fusion Calculus Using HD-Automata. In *CALCO'05*.
6. M. Fiore and S. Staton. Comparing Operational Models of Name-Passing Process Calculi. *Inf. & Comp.*, 204.
7. M. Gabbay and V. Ciancia. Freshness and name-restriction in sets of traces with names. In *FoSSaCS'11*.
8. M. Gabbay and A. Pitts. A new approach to abstract syntax with variable binding. *J. of Formal Aspects of Computing*, 13, 2002.
9. M. J. Gabbay and A. Mathijssen. Nominal (universal) algebra: Equational logic with names and binding. *J. Log. Comput.*, 2009.
10. F. Gadducci, M. Miculan, and U. Montanari. About permutation algebras, (pre)sheaves and named sets. *Higher-Order and Symbolic Computation*, 19, 2006.

11. M. Kaminski and N. Francez. Finite-memory automata. *TCS*, 134, 1994.
12. A. Kurz, T. Suzuki, and E. Tuosto. Nominal monoids. Technical Report CS-10-004, Leicester, 2010.
13. M. Pistore. *History Dependent Automata*. PhD thesis, Dip. di Informatica - Pisa, 1999.
14. A. M. Pitts. Nominal system T. In *POPL'10*.
15. L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL'06*.
16. C. Stirling. Dependency tree automata. In *FoSSaCS'09*.
17. N. Tzevelekos. Fresh-Register Automata. In *POPL'11*.